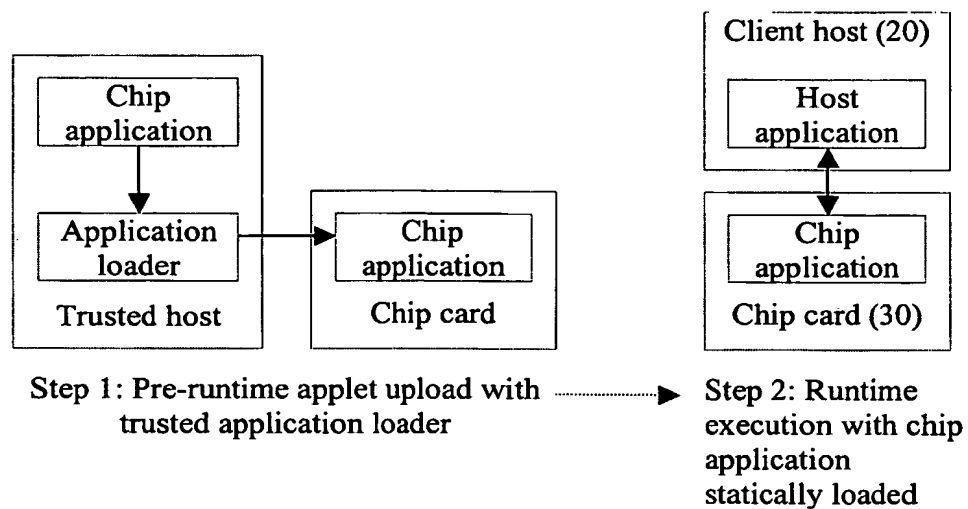


1/10

**Figure 1:** Prior art chip application upload and execution

2/10

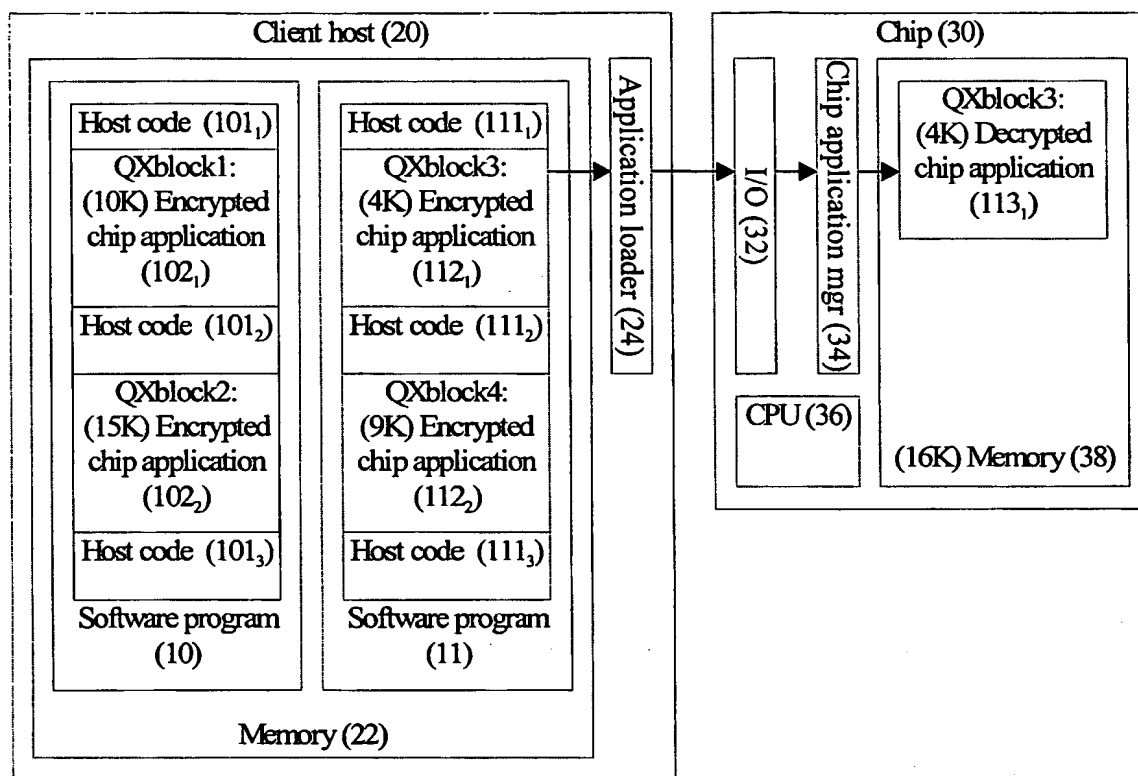


Figure 2: Dynamic upload, decryption and execution of protected chip application blocks (time = t1)

3/10

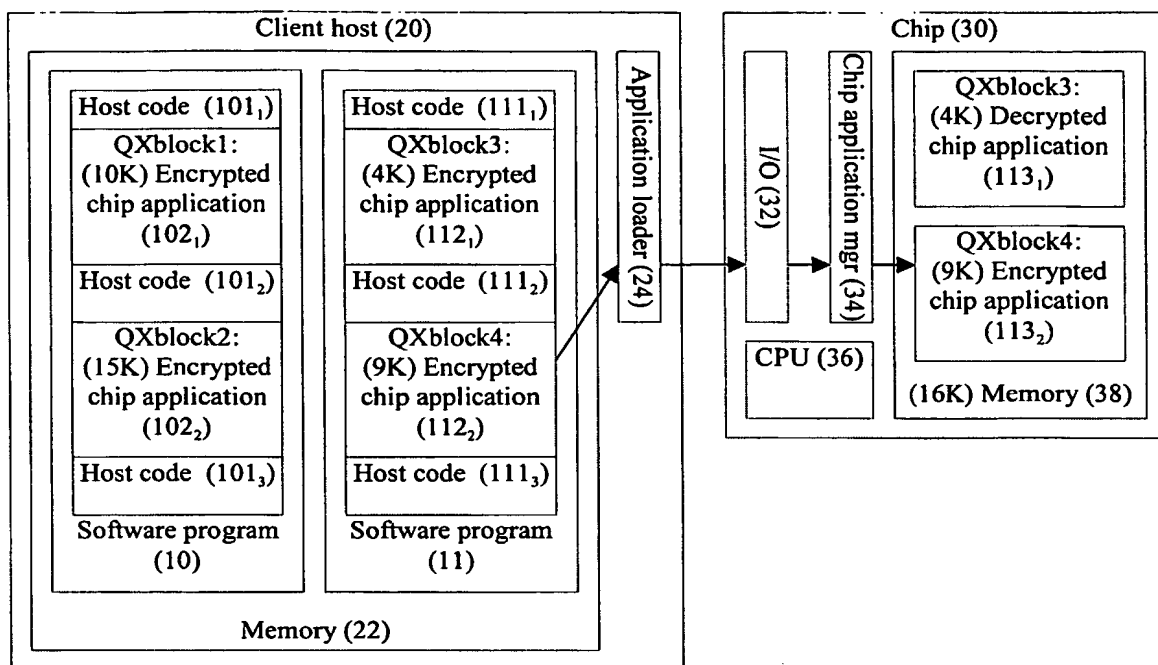


Figure 3: Dynamic upload, decryption and execution of protected chip application blocks (time = t_2)

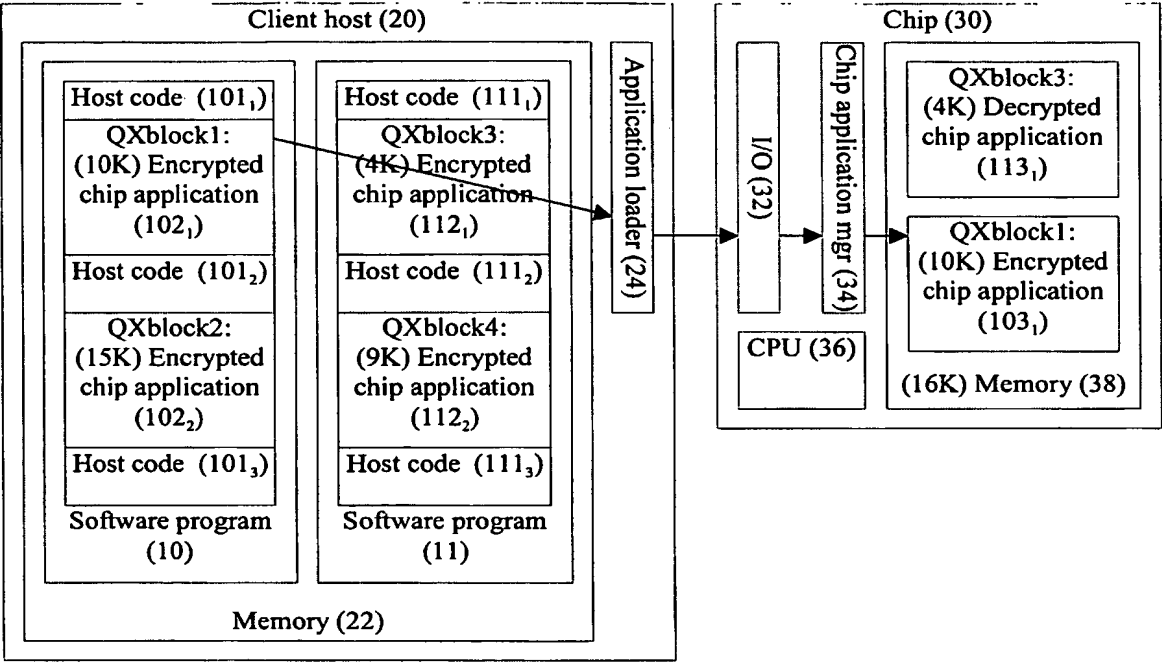


Figure 4: Dynamic upload, decryption and execution of protected chip application blocks (time = t_3)

5/10

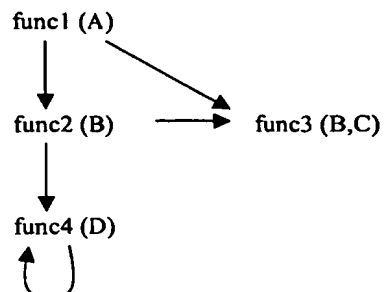


Figure 5: Example: Function call graph of four chip application functions

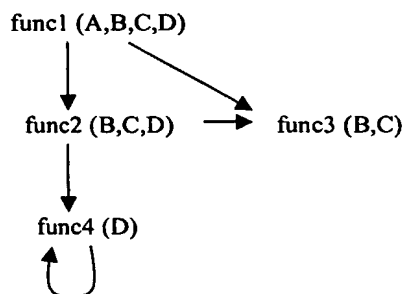


Figure 6: The compiler transitively incorporates variable dependencies into function calls, so that functions may call each other on chip.

6/10

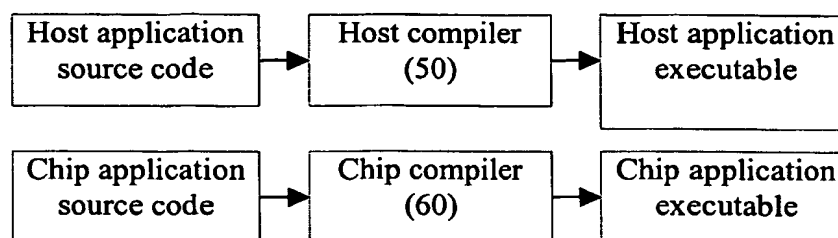


Figure 7: Prior art individual compilation of host and chip applications, using separate compiler development tools

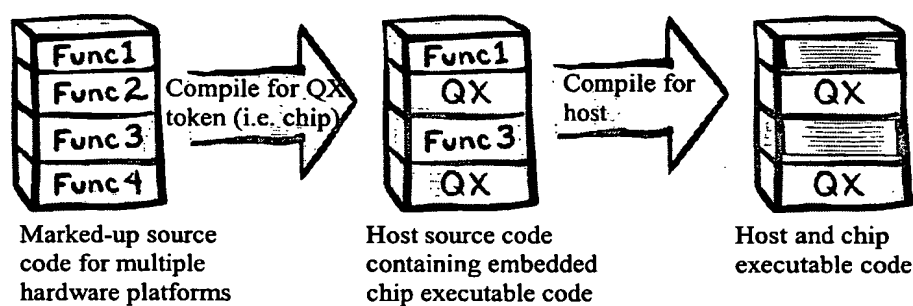


Figure 8: Protection of code; dual-step compilation - one for each processor

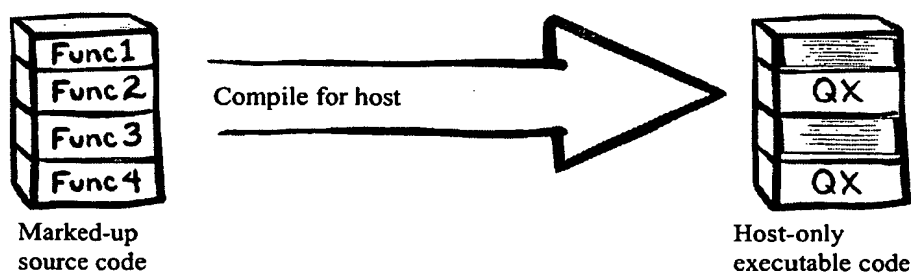


Figure 9: Simulation and debugging; compile directly for host

```
long Power(long base, long exp, long m)
{
    // QXDefault(LicenseId = 65534, ACR=1, Countdown=TRUE)
    long r;

    // QXBegin
    long p;
    short i;
    //QXEnd

    r = 0;

    // QXBegin
    p = 1;
    // calculate base^exp modulus m
    for (i = 1; i <= exp; ++i) {
        p = base * p % m;
    }
    returned = p;
    // QXEnd

    return r;
}

void ComputePower(long nBase, long nExponent, long *nResult)
{
    *nResult = Power(nBase, nExponent, 2100000000);
}
```

Figure 10: Code example: Tagged software application for multiple platforms before compilation

8/10

```

//QXAdded->
#include <string.h>
#include "QXApi.h"

extern const QX_UINT8 QXCode[];

#include <stdio.h>

static void QXCheck (void)
{
    QX_RESULT r = QXGetError ();
    if (r != QX_OK) {
        char text [512];
        QXGetErrorText (r, text, sizeof text);
        printf ("ERROR %d: %s\n", r, text);
    }
}
//<-QXAdded
long Power(long base, long exp, long m)
{
    // qxdefault(LicenseId = 65534, ReaderOptions = SELECT_FIRST)
    long r;

//QXProtected->65534 53689
//6A382777AFACEEDEA9AF376D32A0BE9000F9060F396B6606D77AADA791ED65808
//16FCAA84E087CD8563B3E469FB9F016D671B509C3620DDDE2B6A825B5E06A40
//<-QXProtected

    r = 0;

//QXProtected->65534 10264
//EFFDA14B76859B95EEDB162D76560F1D63A003D37F4B4310B2333ADC97470C01F
//75FA21D9BB08DBAAEC5A09712E6165DFBF564F543CB34F05272C8CC6A292FA73D
//EA632F084FAA46012D6A70B3EE837779F68C3DF99FF4FB18832AE1FFE36C6147B
//D90B5D195DEBB8478A7D1B4315AE70985B9EA68FAB5735F5DCA1A0F72F3755086
//A281FA6AB3C3BEAC8D415470DBB832B0A2F4B3D5DDE473C9241ABA319A064AC78
//F3A31A36E27
//<-QXProtected
//QXAdded->
{
    char QXBuffer [20];
    memcpy (QXBuffer + 0, & base, 4);
    memcpy (QXBuffer + 4, & exp, 4);
    memcpy (QXBuffer + 8, & m, 4);
    memcpy (QXBuffer + 12, & returned, 4);
    QXExecutePtr (65534, 512, QXCode, 0, QXBuffer, 16, QXBuffer, 4);
    memcpy (& r, QXBuffer + 0, 4);
}
    QXCheck ();
//<-QXAdded

    return r;
}

```



```

void ComputePower(long nBase, long nExponent, long *nResult)
{
    *nResult = Power(nBase, nExponent, 2100000000);
}

//QXAdded->
extern const QX_UINT8 QXCode[] = {
0x20,0x56,0x65,0x72,0x73,0x69,0x6F,0x6E,0x20,0x33,0x2E,0x32,0x2D,
0x3E,0x4C,0x09,0x1F,0x3D,0x01,0x00,0x48,0x00,0x20,0x00,0x2A,0xB6,
0xAD,0x2A,0x79,0xE1,0x94,0x1A,0x0A,0x49,0x05,0xAB,0x6D,0x46,0xD2,
0xCF,0xB3,0x1D,0x79,0x26,0x0D,0xBB,0x76,0xEE,0xCD,0x3E,0xA7,0xA1,
0x4D,0x21,0x79,0x5C,0x00,0x00,0x3C,0x00,0x0C,0x04,0x00,0x48,0x00,
0x59,0xA5,0xF8,0x29,0x64,0xE6,0x34,0x62,0xAE,0x95,0xC5,0x69,0x11,
0x66,0x25,0x09,0x94,0x63,0xCA,0x80,0x1B,0x09,0xDD,0x3D,0xB2,0x1B,
0x4C,0x97,0xAE,0x48,0x68,0xB7,0xAD,0x8C,0x25,0x4A,0xA2,0x64,0x8F,
0x16,0x86,0x5D,0x35,0x01,0x23,0xDF,0x32,0x80,0x4C,0xA9,0x52,0x1C,
0x20,0x76,0xD1,0xEB,0xA3,0x5B,0x57,0x23,0x3D,0x00,0x29,0xB0,0x08,
0xAE,0xFC,0x7D,0xD2,0x8B,0xFE,0x71,0x00,0x00,0x00,0x00,0x00,0x00,
0x3C,
};

//<-QXAdded

```

Figure 11: Code example: Software application after compilation, containing encrypted (virtual) machine code for chip platform embedded as source code library function calls.

10/10

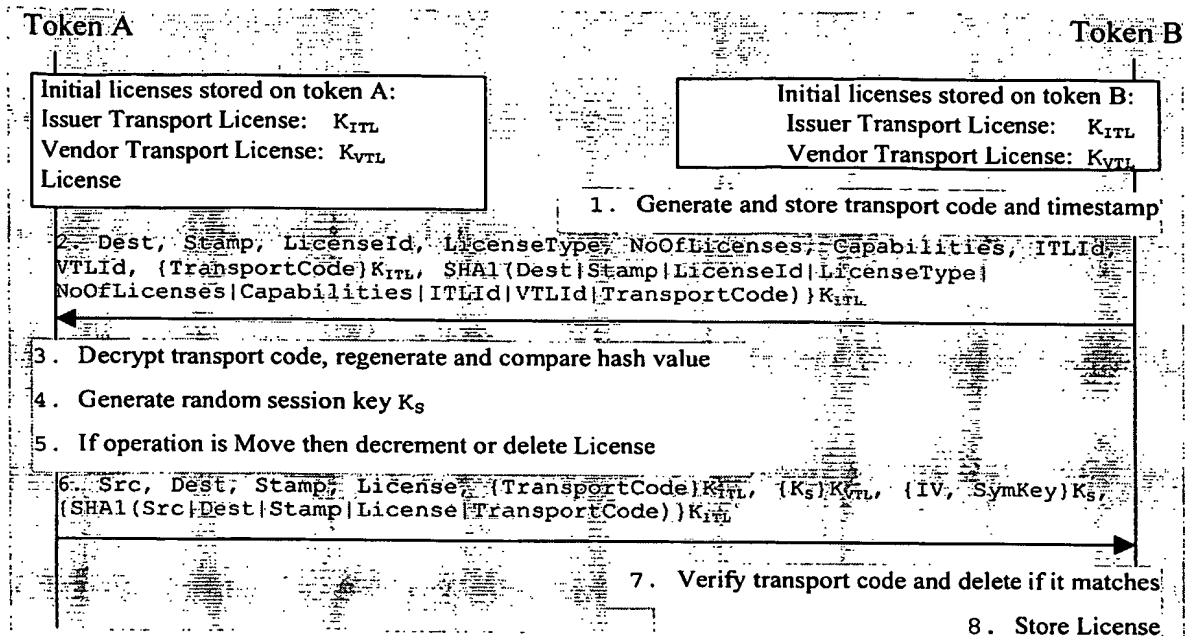


Figure 12: Cryptographic protocol for the transfer of a license from platform A to B, which prevents duplication of license.